Python for the Web and django-fixtureless

Easy Testing in Django

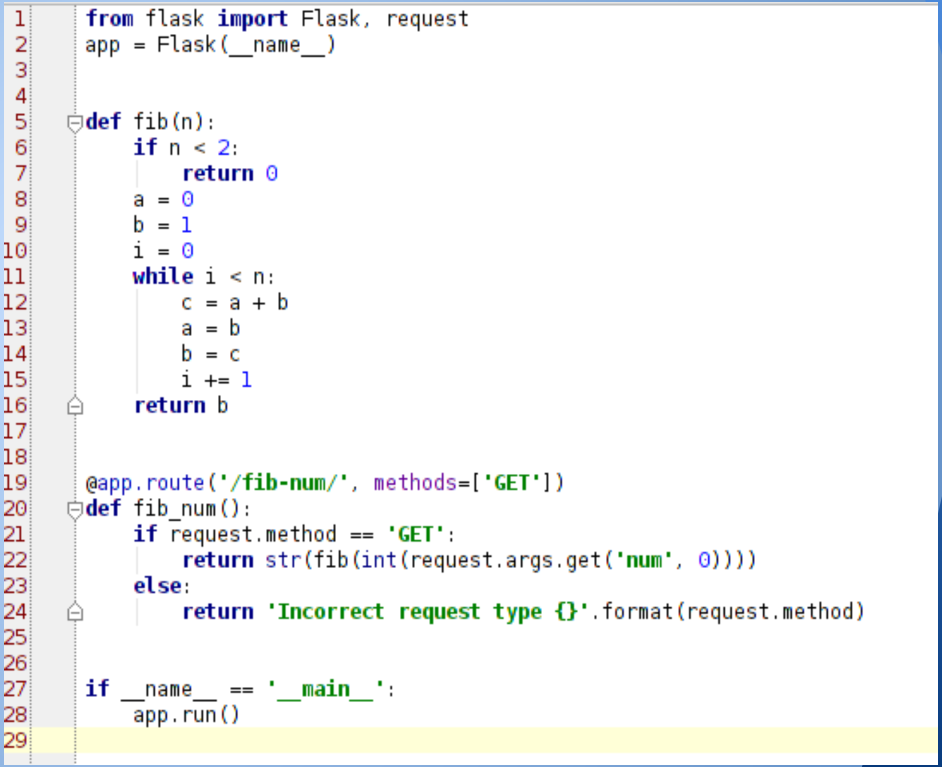
Presented By,
Rico Cordova

Python for the Web

- Zope/Grok
- Flask
- Bottle
- Pylons
- web2py
- Django

Flask

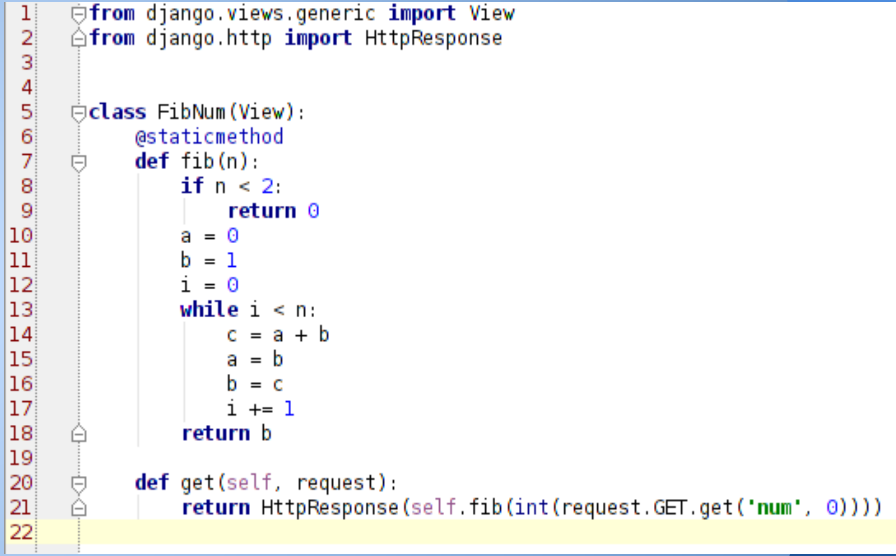
- Lightweight
- Customizable
- Community
- Examples



```python
from flask import Flask, request
app = Flask(__name__)


def fib(n):
    if n < 2:
        return 0
    a = 0
    b = 1
    i = 0
    while i < n:
        c = a + b
        a = b
        b = c
        i += 1
    return b


@app.route('/fib-num/', methods=['GET'])
def fib_num():
    if request.method == 'GET':
        return str(fib(int(request.args.get('num', 0))))
    else:
        return 'Incorrect request type {}'.format(request.method)


if __name__ == '__main__':
    app.run()
```

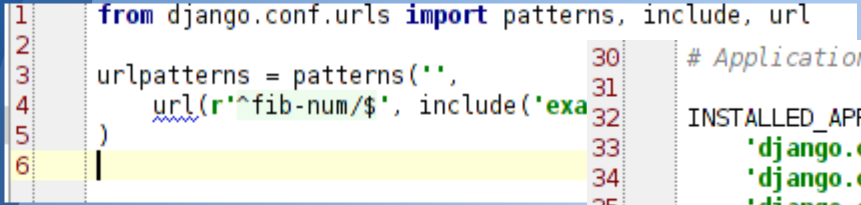
Django

- Extensive Libraries
- Community
- Examples
- Database
- 3rd Party Libraries

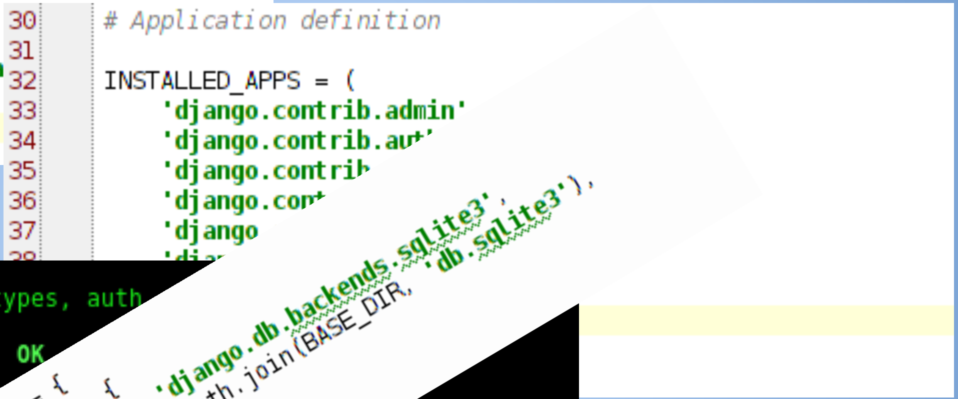


```python
from django.views.generic import View
from django.http import HttpResponse


class FibNum(View):
    @staticmethod
    def fib(n):
        if n < 2:
            return 0
        a = 0
        b = 1
        i = 0
        while i < n:
            c = a + b
            a = b
            b = c
            i += 1
        return b

    def get(self, request):
        return HttpResponse(self.fib(int(request.GET.get('num', 0))))

```

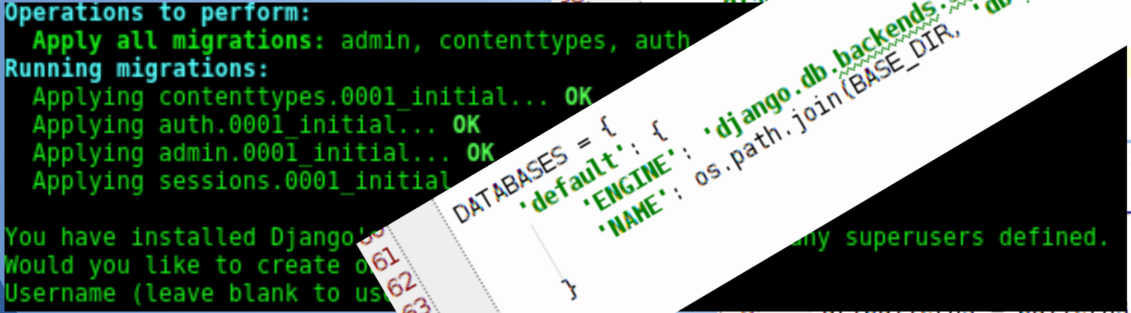
Django Cont…



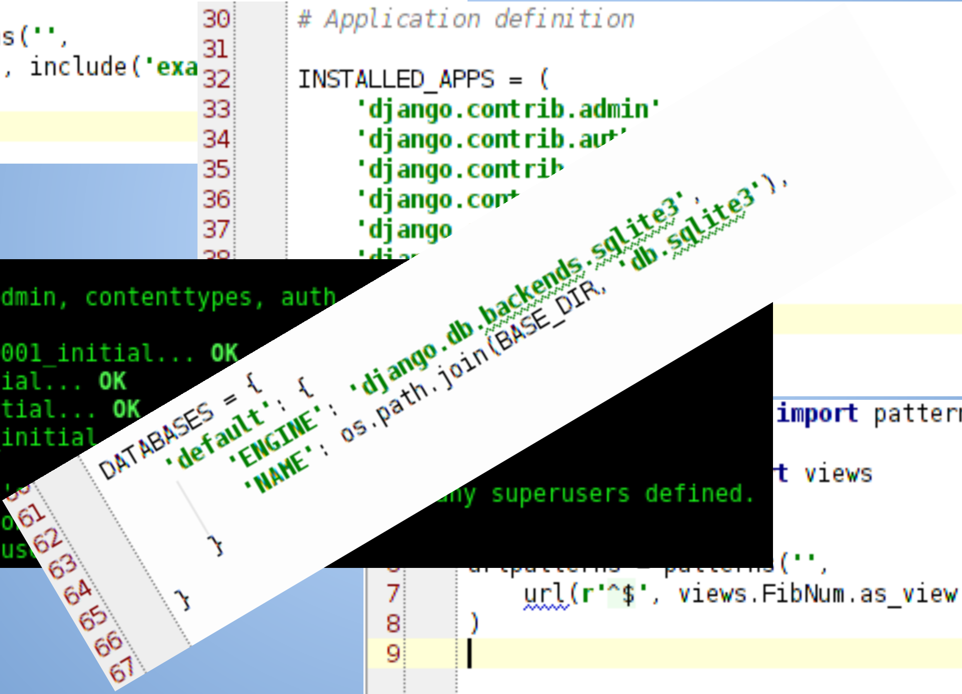
```
from django.conf.urls import patterns, include, url

urlpatterns = patterns('',
    url(r'^fib-num/$', include('exa
)
```



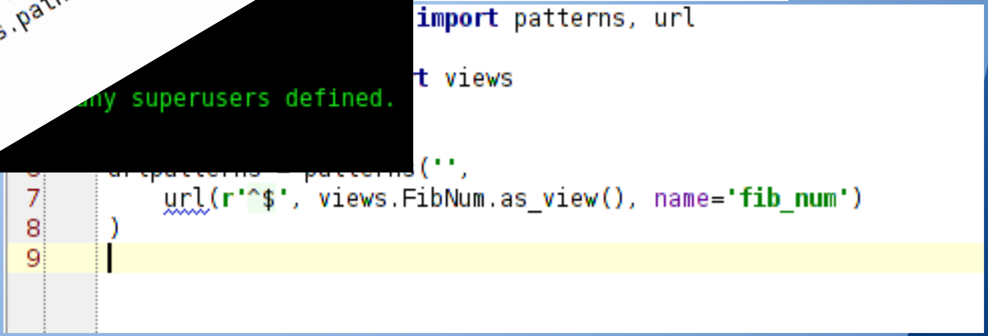
```
# Application definition

INSTALLED_APPS = (
    'django.contrib.admin'
    'django.contrib.aut
    'django.contrib
    'django.cont
    'django
    'dja
```



```
Operations to perform:
  Apply all migrations: admin, contenttypes, auth
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial

You have installed Django's
Would you like to create o
Username (leave blank to us
ny superusers defined.
```



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

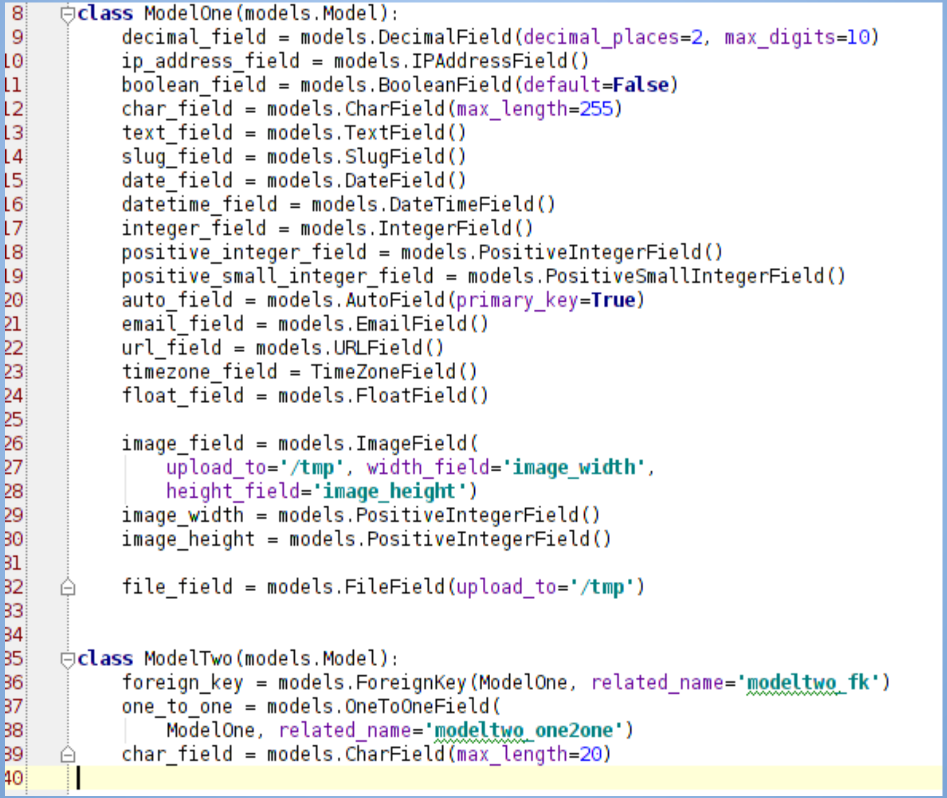


```
import patterns, url
t views

urlpatterns = patterns('',
    url(r'^$', views.FibNum.as_view(), name='fib_num')
)
```

Why Use Django?

- Available Libraries
- DRY instead of DIY
- Object Relational Mapping (ORM)

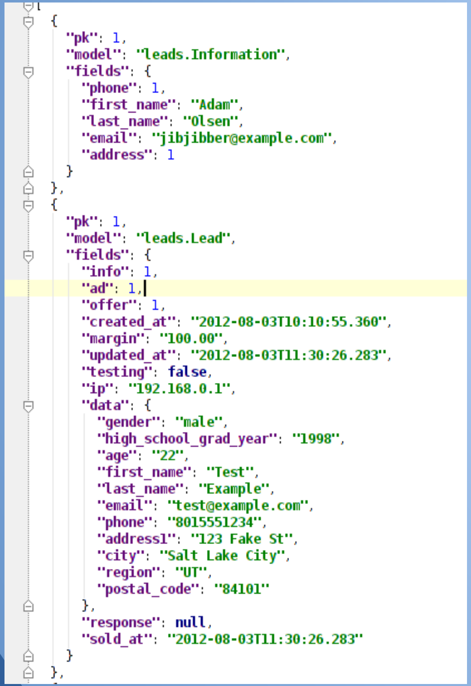
Data Model = Database Schema



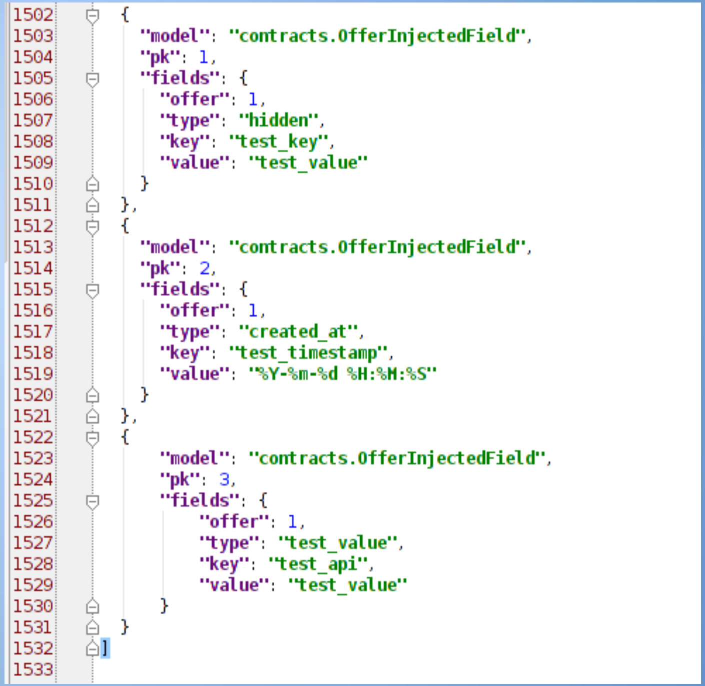
```
class ModelOne(models.Model):
    decimal_field = models.DecimalField(decimal_places=2, max_digits=10)
    ip_address_field = models.IPAddressField()
    boolean_field = models.BooleanField(default=False)
    char_field = models.CharField(max_length=255)
    text_field = models.TextField()
    slug_field = models.SlugField()
    date_field = models.DateField()
    datetime_field = models.DateTimeField()
    integer_field = models.IntegerField()
    positive_integer_field = models.PositiveIntegerField()
    positive_small_integer_field = models.PositiveSmallIntegerField()
    auto_field = models.AutoField(primary_key=True)
    email_field = models.EmailField()
    url_field = models.URLField()
    timezone_field = TimeZoneField()
    float_field = models.FloatField()

    image_field = models.ImageField(
        upload_to='/tmp', width_field='image_width',
        height_field='image_height')
    image_width = models.PositiveIntegerField()
    image_height = models.PositiveIntegerField()

    file_field = models.FileField(upload_to='/tmp')


class ModelTwo(models.Model):
    foreign_key = models.ForeignKey(ModelOne, related_name='modeltwo_fk')
    one_to_one = models.OneToOneField(
        ModelOne, related_name='modeltwo_one2one')
    char_field = models.CharField(max_length=20)

```

So Much Data!

Enter Fixtures



```
[
  {
    "pk": 1,
    "model": "leads.Information",
    "fields": {
      "phone": 1,
      "first_name": "Adam",
      "last_name": "Olsen",
      "email": "jibjibber@example.com",
      "address": 1
    }
  },
  {
    "pk": 1,
    "model": "leads.Lead",
    "fields": {
      "info": 1,
      "ad": 1,
      "offer": 1,
      "created_at": "2012-08-03T10:10:55.360",
      "margin": "100.00",
      "updated_at": "2012-08-03T11:30:26.283",
      "testing": false,
      "ip": "192.168.0.1",
      "data": {
        "gender": "male",
        "high_school_grad_year": "1998",
        "age": "22",
        "first_name": "Test",
        "last_name": "Example",
        "email": "test@example.com",
        "phone": "8015551234",
        "address1": "123 Fake St",
        "city": "Salt Lake City",
        "region": "UT",
        "postal_code": "84101"
      },
      "response": null,
      "sold_at": "2012-08-03T11:30:26.283"
    }
  },
```



```
  {
    "model": "contracts.OfferInjectedField",
    "pk": 1,
    "fields": {
      "offer": 1,
      "type": "hidden",
      "key": "test_key",
      "value": "test_value"
    }
  },
  {
    "model": "contracts.OfferInjectedField",
    "pk": 2,
    "fields": {
      "offer": 1,
      "type": "created_at",
      "key": "test_timestamp",
      "value": "%Y-%m-%d %H:%M:%S"
    }
  },
  {
      "model": "contracts.OfferInjectedField",
      "pk": 3,
      "fields": {
          "offer": 1,
          "type": "test_value",
          "key": "test_api",
          "value": "test_value"
      }
  }
]
```

When the Data Model Changes

Enter Fixtureless

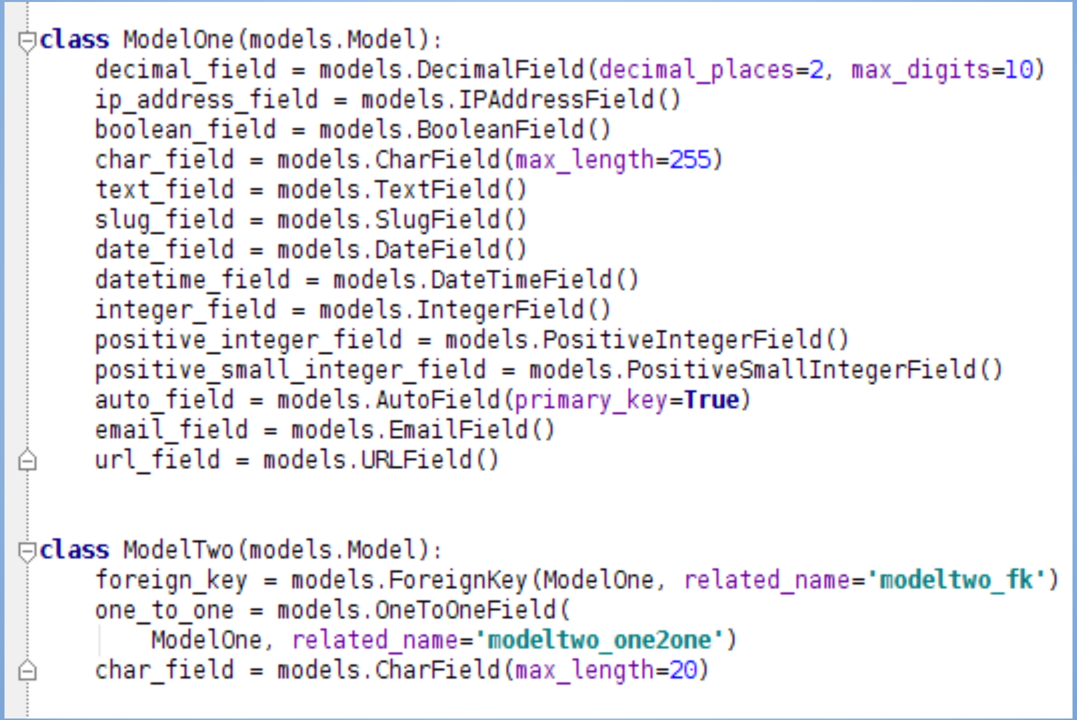
django-fixtureless

- Create test objects
- No more tedious upkeep of fixtures
- Handles auto-generation of complex relationships

Simple to Use

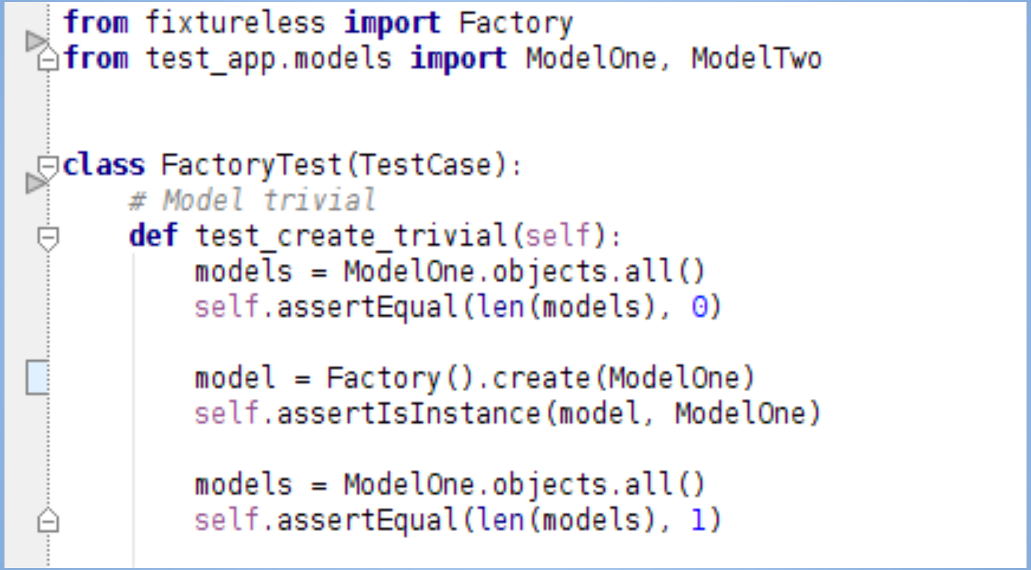
- build():
  - generates a Django model object (or list of objects)
- create():
  - similar to build but the object gets saved to the database

Your Django Models (Again)

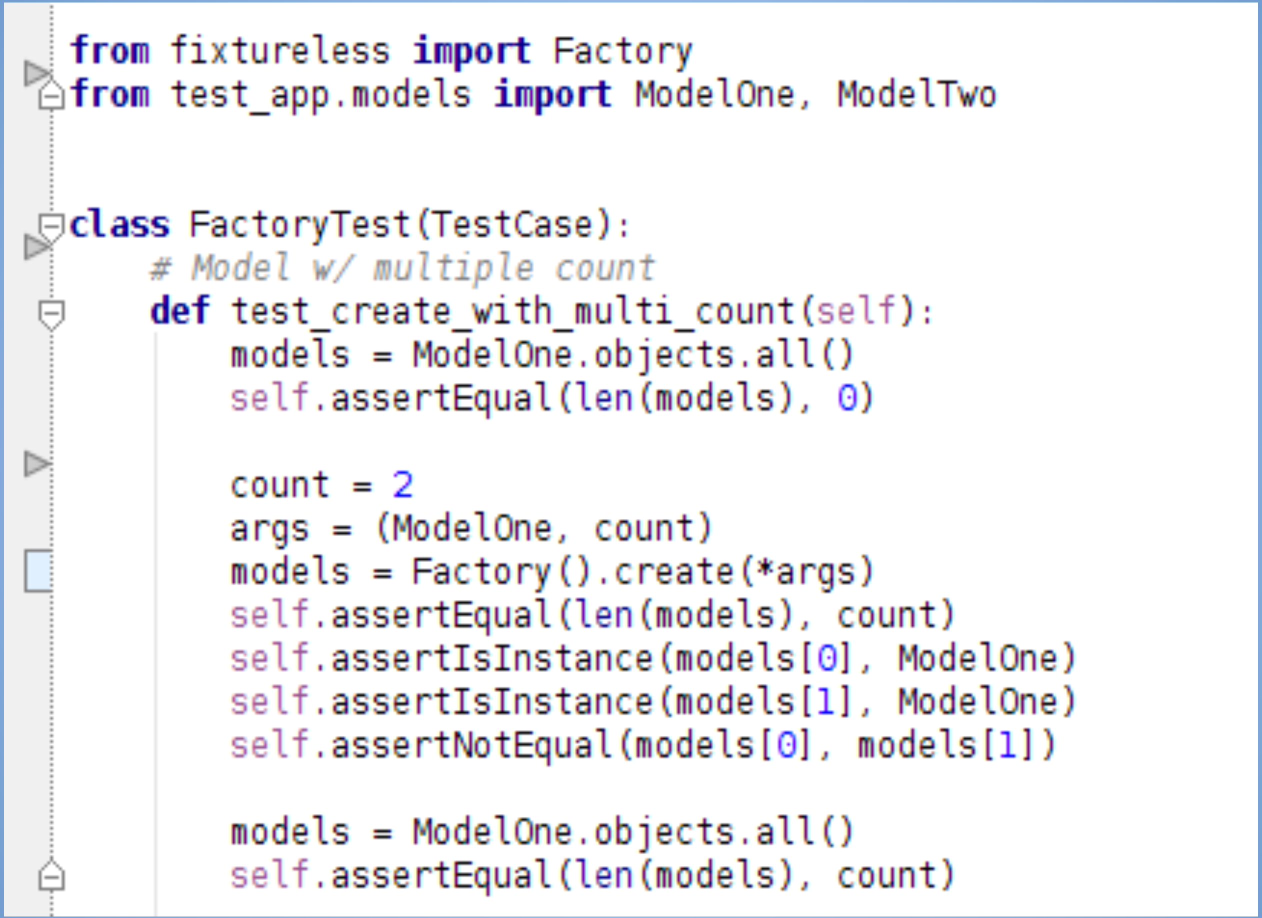


```python
class ModelOne(models.Model):
    decimal_field = models.DecimalField(decimal_places=2, max_digits=10)
    ip_address_field = models.IPAddressField()
    boolean_field = models.BooleanField()
    char_field = models.CharField(max_length=255)
    text_field = models.TextField()
    slug_field = models.SlugField()
    date_field = models.DateField()
    datetime_field = models.DateTimeField()
    integer_field = models.IntegerField()
    positive_integer_field = models.PositiveIntegerField()
    positive_small_integer_field = models.PositiveSmallIntegerField()
    auto_field = models.AutoField(primary_key=True)
    email_field = models.EmailField()
    url_field = models.URLField()


class ModelTwo(models.Model):
    foreign_key = models.ForeignKey(ModelOne, related_name='modeltwo_fk')
    one_to_one = models.OneToOneField(
        ModelOne, related_name='modeltwo_one2one')
    char_field = models.CharField(max_length=20)
```

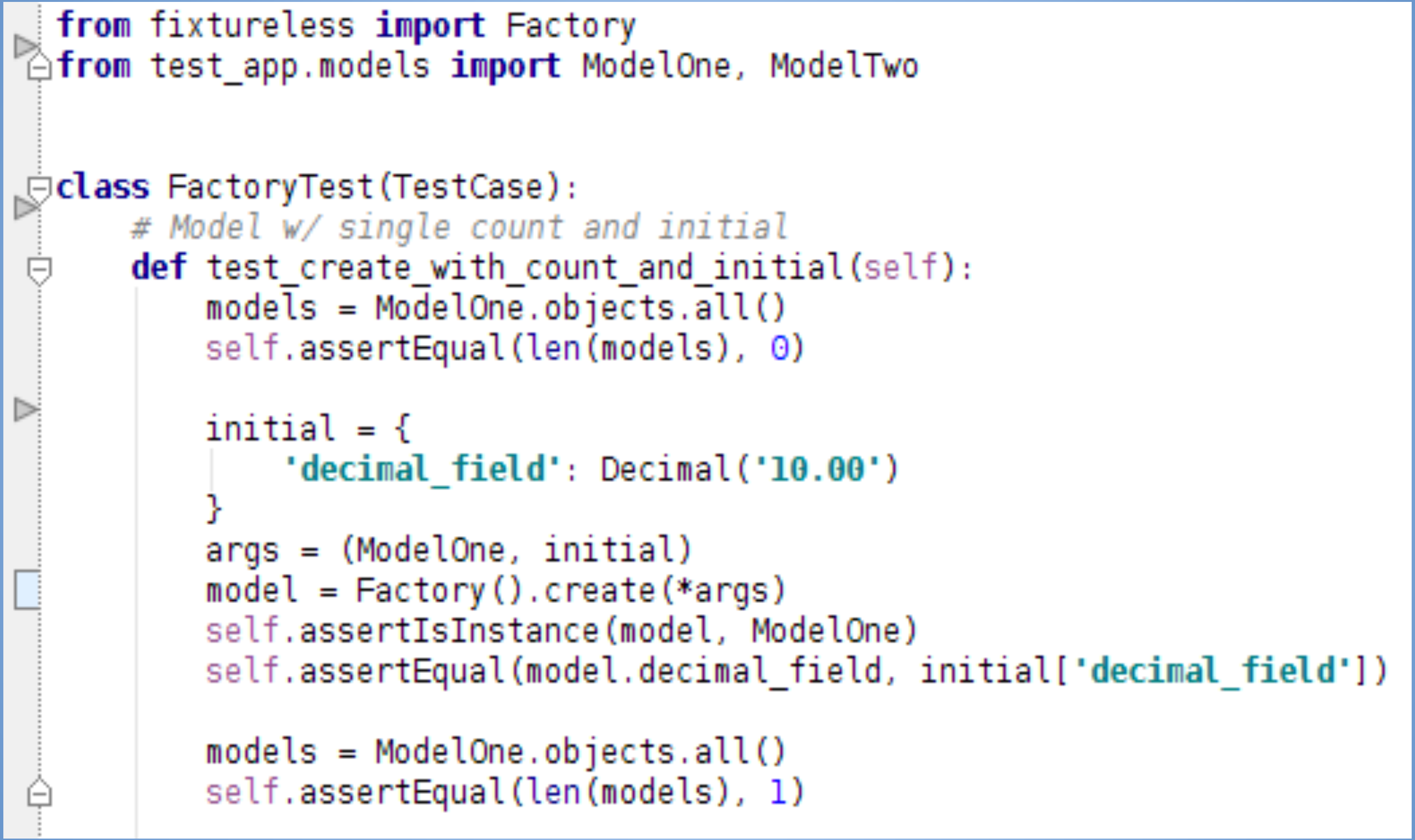
Use Cases for create()



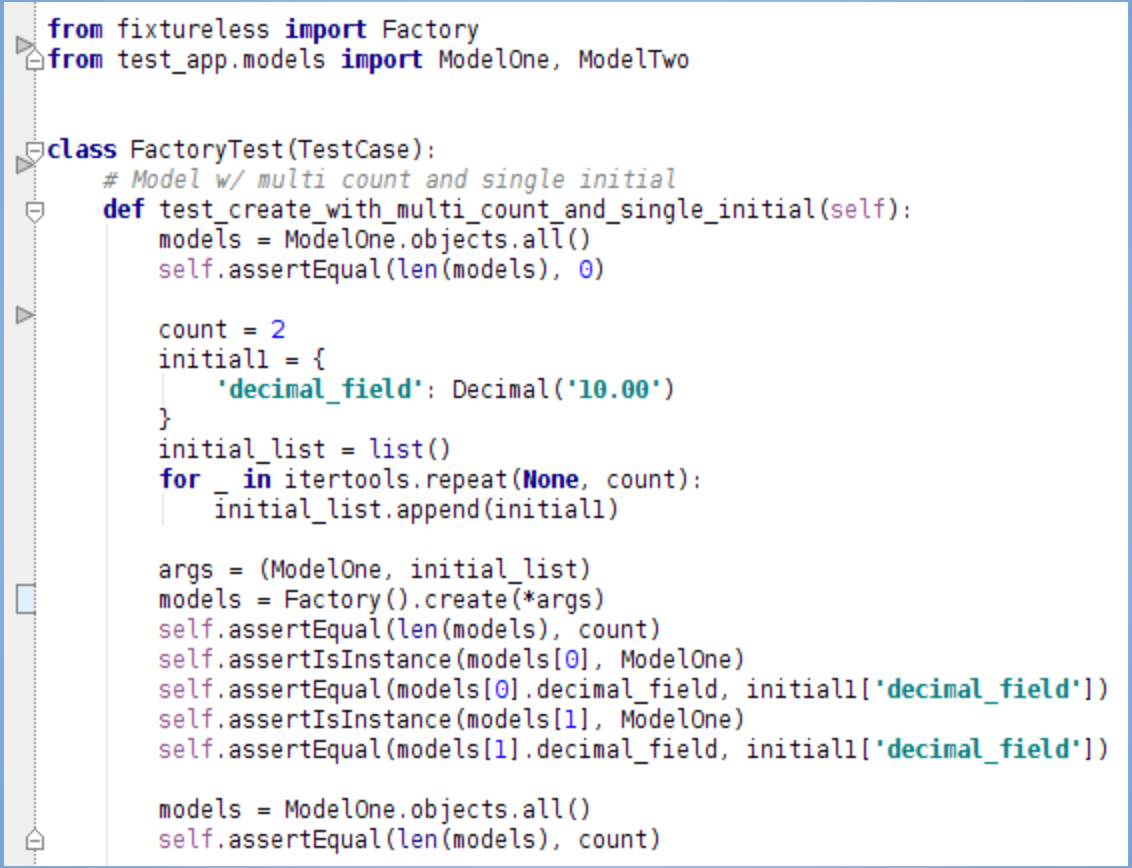
```
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Model trivial
    def test_create_trivial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)

        model = Factory().create(ModelOne)
        self.assertIsInstance(model, ModelOne)

        models = ModelOne.objects.all()
        self.assertEqual(len(models), 1)
```



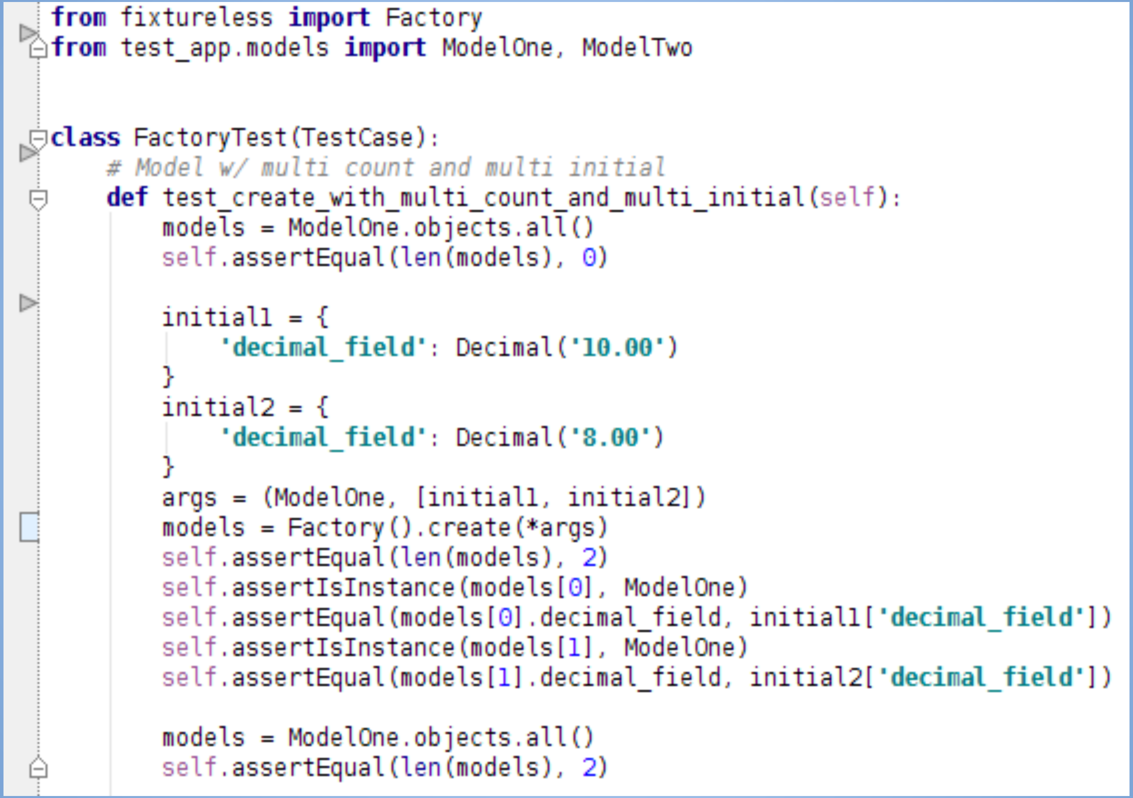
```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Model w/ multiple count
    def test_create_with_multi_count(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)

        count = 2
        args = (ModelOne, count)
        models = Factory().create(*args)
        self.assertEqual(len(models), count)
        self.assertIsInstance(models[0], ModelOne)
        self.assertIsInstance(models[1], ModelOne)
        self.assertNotEqual(models[0], models[1])

        models = ModelOne.objects.all()
        self.assertEqual(len(models), count)
```



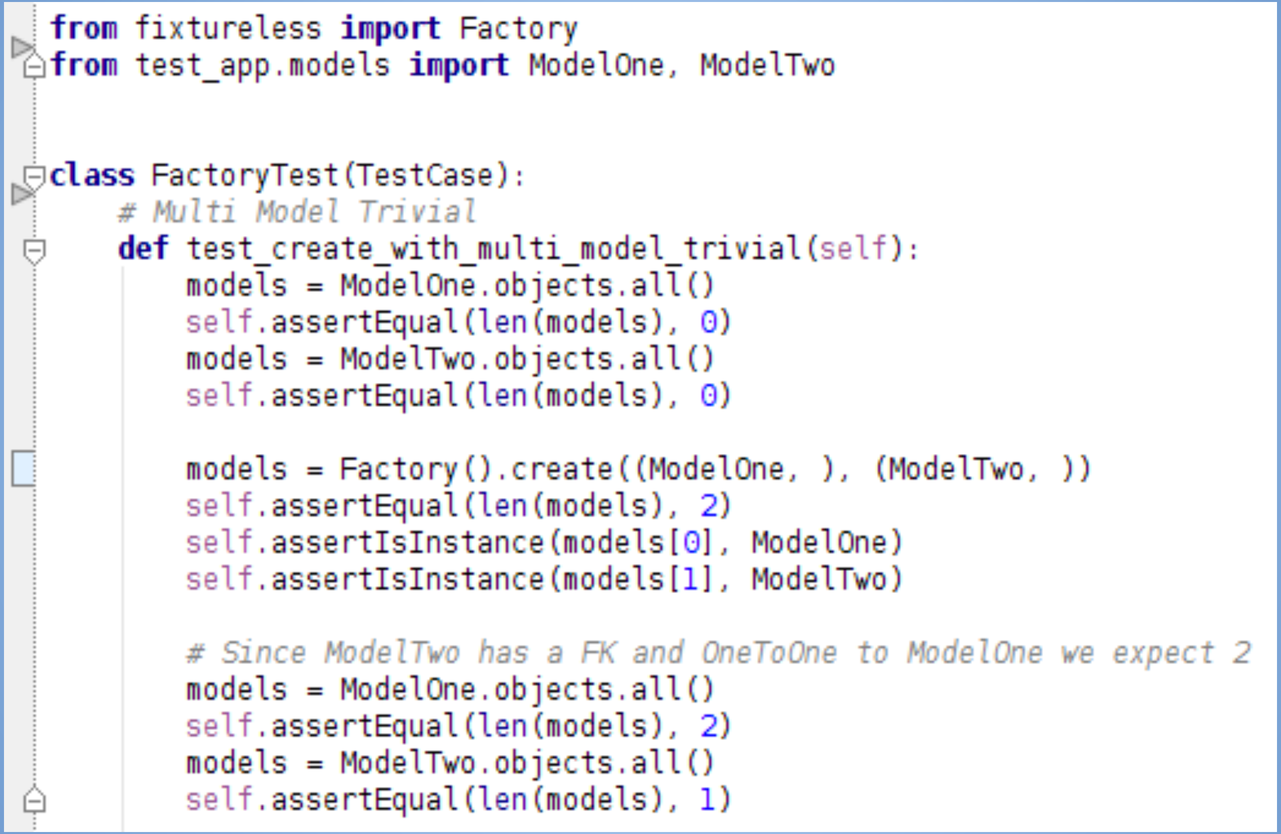
```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Model w/ single count and initial
    def test_create_with_count_and_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)

        initial = {
            'decimal_field': Decimal('10.00')
        }
        args = (ModelOne, initial)
        model = Factory().create(*args)
        self.assertIsInstance(model, ModelOne)
        self.assertEqual(model.decimal_field, initial['decimal_field'])

        models = ModelOne.objects.all()
        self.assertEqual(len(models), 1)
```



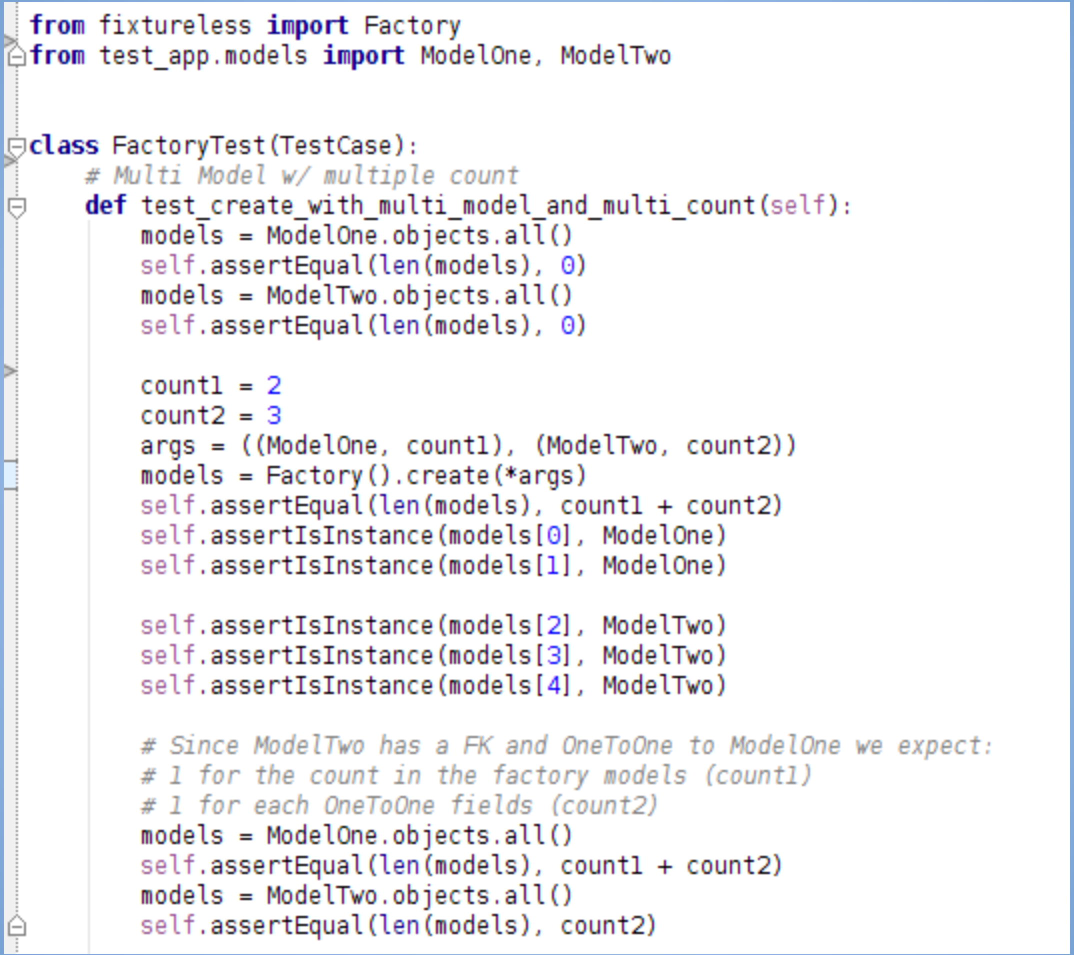
```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Model w/ multi count and single initial
    def test_create_with_multi_count_and_single_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)

        count = 2
        initial1 = {
            'decimal_field': Decimal('10.00')
        }
        initial_list = list()
        for _ in itertools.repeat(None, count):
            initial_list.append(initial1)

        args = (ModelOne, initial_list)
        models = Factory().create(*args)
        self.assertEqual(len(models), count)
        self.assertIsInstance(models[0], ModelOne)
        self.assertEqual(models[0].decimal_field, initial1['decimal_field'])
        self.assertIsInstance(models[1], ModelOne)
        self.assertEqual(models[1].decimal_field, initial1['decimal_field'])

        models = ModelOne.objects.all()
        self.assertEqual(len(models), count)
```



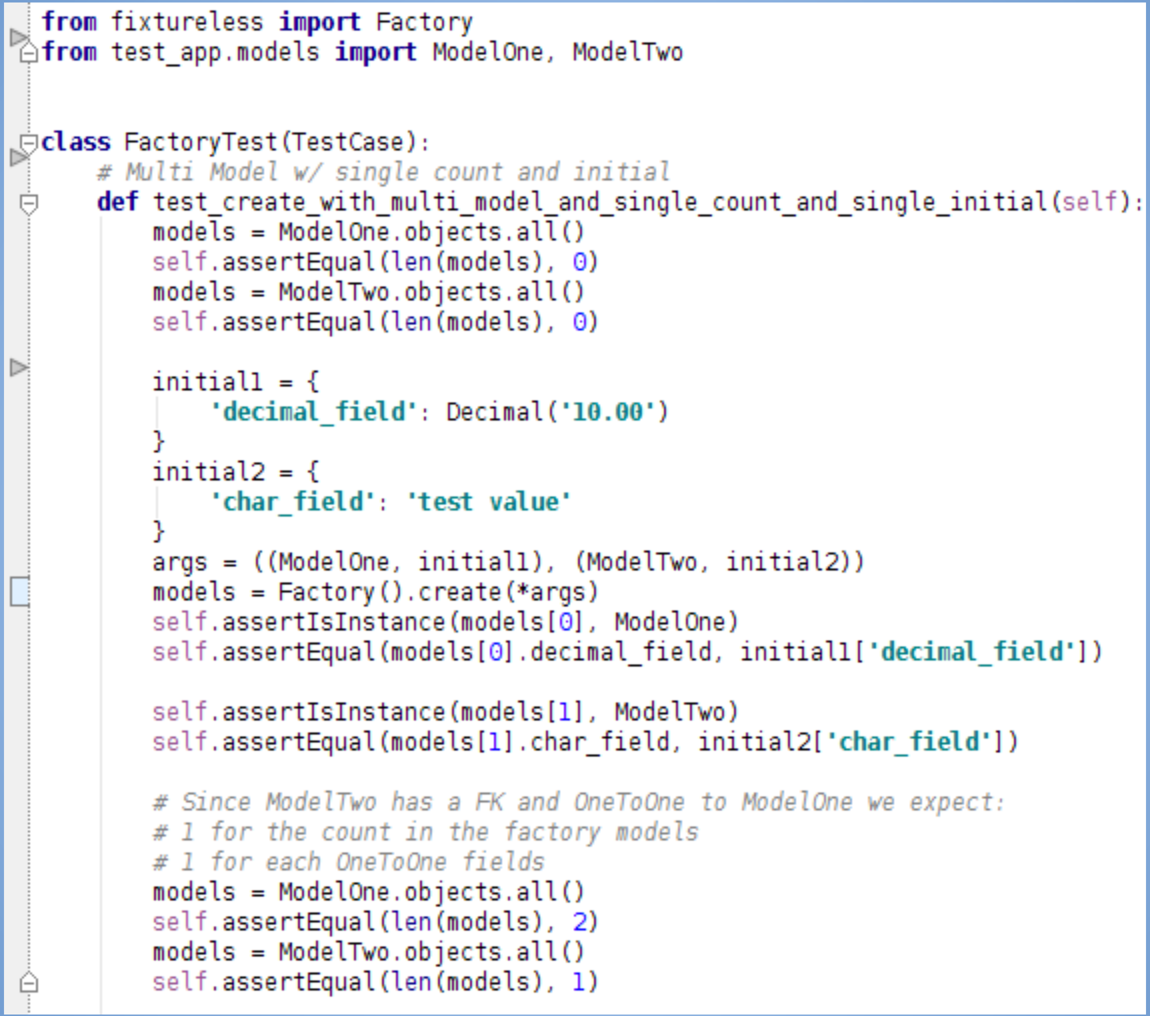
```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Model w/ multi count and multi initial
    def test_create_with_multi_count_and_multi_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)

        initial1 = {
            'decimal_field': Decimal('10.00')
        }
        initial2 = {
            'decimal_field': Decimal('8.00')
        }
        args = (ModelOne, [initial1, initial2])
        models = Factory().create(*args)
        self.assertEqual(len(models), 2)
        self.assertIsInstance(models[0], ModelOne)
        self.assertEqual(models[0].decimal_field, initial1['decimal_field'])
        self.assertIsInstance(models[1], ModelOne)
        self.assertEqual(models[1].decimal_field, initial2['decimal_field'])

        models = ModelOne.objects.all()
        self.assertEqual(len(models), 2)
```



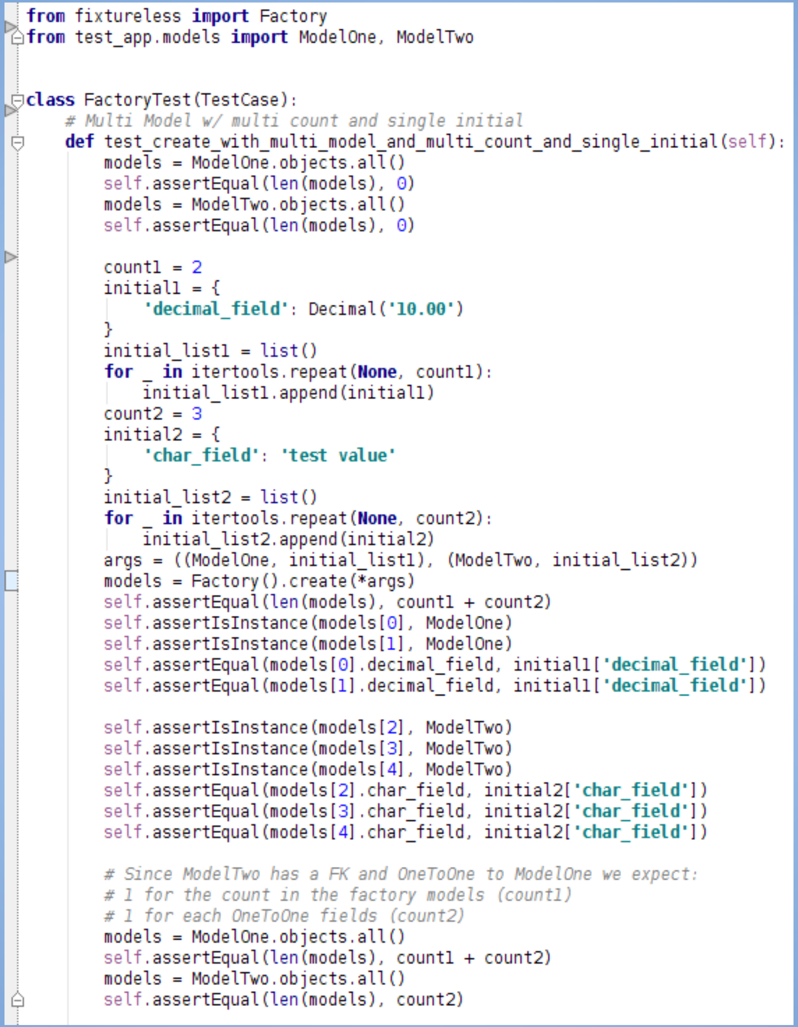
```
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Multi Model Trivial
    def test_create_with_multi_model_trivial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 0)

        models = Factory().create((ModelOne, ), (ModelTwo, ))
        self.assertEqual(len(models), 2)
        self.assertIsInstance(models[0], ModelOne)
        self.assertIsInstance(models[1], ModelTwo)

        # Since ModelTwo has a FK and OneToOne to ModelOne we expect 2
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 2)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 1)
```



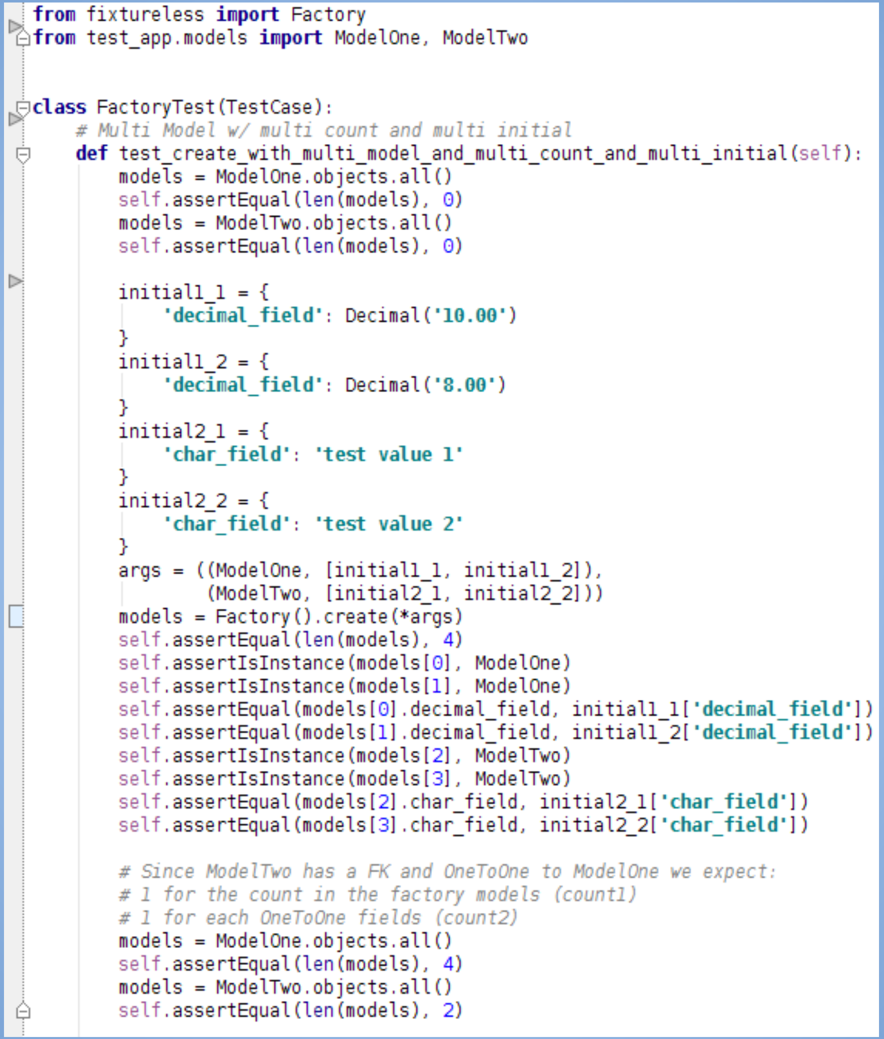
```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Multi Model w/ multiple count
    def test_create_with_multi_model_and_multi_count(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 0)

        count1 = 2
        count2 = 3
        args = ((ModelOne, count1), (ModelTwo, count2))
        models = Factory().create(*args)
        self.assertEqual(len(models), count1 + count2)
        self.assertIsInstance(models[0], ModelOne)
        self.assertIsInstance(models[1], ModelOne)

        self.assertIsInstance(models[2], ModelTwo)
        self.assertIsInstance(models[3], ModelTwo)
        self.assertIsInstance(models[4], ModelTwo)

        # Since ModelTwo has a FK and OneToOne to ModelOne we expect:
        # 1 for the count in the factory models (count1)
        # 1 for each OneToOne fields (count2)
        models = ModelOne.objects.all()
        self.assertEqual(len(models), count1 + count2)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), count2)
```



```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Multi Model w/ single count and initial
    def test_create_with_multi_model_and_single_count_and_single_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 0)

        initial1 = {
            'decimal_field': Decimal('10.00')
        }
        initial2 = {
            'char_field': 'test value'
        }
        args = ((ModelOne, initial1), (ModelTwo, initial2))
        models = Factory().create(*args)
        self.assertIsInstance(models[0], ModelOne)
        self.assertEqual(models[0].decimal_field, initial1['decimal_field'])

        self.assertIsInstance(models[1], ModelTwo)
        self.assertEqual(models[1].char_field, initial2['char_field'])

        # Since ModelTwo has a FK and OneToOne to ModelOne we expect:
        # 1 for the count in the factory models
        # 1 for each OneToOne fields
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 2)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 1)
```



```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Multi Model w/ multi count and single initial
    def test_create_with_multi_model_and_multi_count_and_single_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 0)

        count1 = 2
        initial1 = {
            'decimal_field': Decimal('10.00')
        }
        initial_list1 = list()
        for _ in itertools.repeat(None, count1):
            initial_list1.append(initial1)
        count2 = 3
        initial2 = {
            'char_field': 'test value'
        }
        initial_list2 = list()
        for _ in itertools.repeat(None, count2):
            initial_list2.append(initial2)
        args = ((ModelOne, initial_list1), (ModelTwo, initial_list2))
        models = Factory().create(*args)
        self.assertEqual(len(models), count1 + count2)
        self.assertIsInstance(models[0], ModelOne)
        self.assertIsInstance(models[1], ModelOne)
        self.assertEqual(models[0].decimal_field, initial1['decimal_field'])
        self.assertEqual(models[1].decimal_field, initial1['decimal_field'])

        self.assertIsInstance(models[2], ModelTwo)
        self.assertIsInstance(models[3], ModelTwo)
        self.assertIsInstance(models[4], ModelTwo)
        self.assertEqual(models[2].char_field, initial2['char_field'])
        self.assertEqual(models[3].char_field, initial2['char_field'])
        self.assertEqual(models[4].char_field, initial2['char_field'])

        # Since ModelTwo has a FK and OneToOne to ModelOne we expect:
        # 1 for the count in the factory models (count1)
        # 1 for each OneToOne fields (count2)
        models = ModelOne.objects.all()
        self.assertEqual(len(models), count1 + count2)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), count2)
```



```python
from fixtureless import Factory
from test_app.models import ModelOne, ModelTwo


class FactoryTest(TestCase):
    # Multi Model w/ multi count and multi initial
    def test_create_with_multi_model_and_multi_count_and_multi_initial(self):
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 0)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 0)

        initial1_1 = {
            'decimal_field': Decimal('10.00')
        }
        initial1_2 = {
            'decimal_field': Decimal('8.00')
        }
        initial2_1 = {
            'char_field': 'test value 1'
        }
        initial2_2 = {
            'char_field': 'test value 2'
        }
        args = ((ModelOne, [initial1_1, initial1_2]),
                (ModelTwo, [initial2_1, initial2_2]))
        models = Factory().create(*args)
        self.assertEqual(len(models), 4)
        self.assertIsInstance(models[0], ModelOne)
        self.assertIsInstance(models[1], ModelOne)
        self.assertEqual(models[0].decimal_field, initial1_1['decimal_field'])
        self.assertEqual(models[1].decimal_field, initial1_2['decimal_field'])
        self.assertIsInstance(models[2], ModelTwo)
        self.assertIsInstance(models[3], ModelTwo)
        self.assertEqual(models[2].char_field, initial2_1['char_field'])
        self.assertEqual(models[3].char_field, initial2_2['char_field'])

        # Since ModelTwo has a FK and OneToOne to ModelOne we expect:
        # 1 for the count in the factory models (count1)
        # 1 for each OneToOne fields (count2)
        models = ModelOne.objects.all()
        self.assertEqual(len(models), 4)
        models = ModelTwo.objects.all()
        self.assertEqual(len(models), 2)
```

django-fixtureless

Available on PYPI

```
pip install django-fixtureless
```

Fork from Github

https://www.github.com/ricomoss/django-fixtureless